AMATH 482/582: HOMEWORK 3

KATIE HIPPE

AMATH Department, University of Washington, Seattle, WA
kahippe@uw.edu

Abstract. The MNIST data set contains images of handwritten digits from 0-9, which can be analyzed and sorted according to various classification algorithms. We have used PCA modes to project these images into a lower-dimension space and created an algorithm that will recognize the correct digit contained in each image, then cross-validated and tested the algorithm for accuracy.

1. Introduction and Overview

Handwriting can differ immensely from person to person, making it difficult for even the human eye to distinguish the writer's intended letter or number. The MNIST data set contains examples of various writer's versions of digits 0-9, all labeled with the intended number. For each image, we wish to classify it as the correct digit. However, these images are dense and contain a lot of unnecessary information, as each image is 28×28 pixels for a total of 784 separate pixels in each image.

We wish to shrink this data into lower dimensions while preserving a majority of the information, then classify this shrunken data into the correct digit. We can do this with a variety of classifiers, for the purposes of this paper focusing on Ridge Regression, KNN, and SVM as an extra comparison.

We first analyzed the data using PCA to find the lowest possible number of modes to preserve most of the data. Using this lowered-dimensional data, we set up and evaluated each classifier for optimal input parameters. We compared each classifier's performance on our training and testing data sets, using cross-validation to improve our trust in each classifier.

2. Theoretical Background

We first obtained 60000 samples of handwritten digits with which to train our classifiers, all displayed in 28×28 matrices of gray-scale pixels. We also got 10000 samples to test our classifiers. Thus we had a training matrix with dimensions 60000×784, after flattening each image, a higher dimension than we would prefer to work with.

To reduce the dimensionality of our data, we conducted a Principal Component Analysis (PCA) to find the most important components of our data. To do so, we used Singular Value Decomposition (SVD) (1):

$$X = U\Sigma V^T \tag{1}$$

where X is our training data matrix. It has been decomposed into three component matrices: U, the left singular vectors, V, the right singular vectors, and Σ, a diagonal matrix filled with the singular values of X sorted in descending order.

Once we have found U using the entirety of our training data, we may use it to project a given data sample z (all 784 pixels) onto the determined PC modes:

$$z = U^T z$$

Date: March 3, 2025.

where we have as many PC modes as pixels in our initial data. However, in the case of our data this means 784 PC modes, and we wished to reduce the dimensions of our data. So instead we may project on $k-$PC modes, so instead of 784 modes we only have k, using (2)

$$z_k = U_k^T z \tag{2}$$

where U_k^T has the first k column vectors kept intact while the column vectors from $k+1$ and onward are set to 0. When we apply this to a data sample z, the result will be a reduced-dimension in $k-$PCA space, which we may compare with other data in the same $k-$PCA space.

The cumulative energy may then be calculated by the cumulative sum of the variance as explained by each principal component. These are found by dividing the cumulative sum of eigenvalues up to the current number of principal components by the sum of all eigenvalues:

$$E_i = \frac{\sqrt{\sum_{n=1}^{i} \Sigma_{n,n}}}{\sqrt{\sum_{n=1}^{N} \Sigma_{n,n}}} \tag{3}$$

where E_i is the cumulative energy for i components, N is the total number of possible components (in this case, N = 784), and we're using the Σ matrix calculated from 1, as it contains the squared eigenvalues of the principal components. This is equivalent to taking the Frobenius norm of the reconstructed matrix.

To classify our data, we first used a Ridge Regression classifier (4). This model performs a linear regression including a regularization parameter.

$$\beta = \text{argmin}_{\beta \in \mathbb{R}^J} \frac{1}{2\sigma^2} ||X\beta - Y||^2 + \frac{\lambda}{2}||\beta||_2^2 \tag{4}$$

In this case, we are conducting a more general polynomial regression to find the parameter vector β that best transforms the data matrix X into the label matrix Y, that is, minimize the error over all β possibilities. The regularization parameter $\frac{\lambda}{2}||\beta||_2^2$ is what specifically defines this process as Ridge Regression. This parameter helps when X is nearly linearly dependent. The specific value of λ is determined via cross-validation using MSE (see 3 for more detail).

We also used K Nearest Neighbors, or KNN. In this scheme, we specify the number of neighbors to consider, for example $k = 3$. Then for each data point, we will consider the 3 nearest neighbors, as defined by Euclidean distance, and their labels. Our data point is then classified by majority vote of the labels of those nearest neighbors. Each data point and their specific neighbors are considered until every one is classified.

Finally, we considered (RBF) Support Vector Machines, or SVM. This algorithm works by finding a hyperplane to act as our decision boundary to separate our different data points. In a high-dimensional data set, there is a margin between different classification types, which is larger for well-separated data. SVM will find the hyperplane to use that maximizes this margin, so we may have a greater degree of trust in its classifications.

3. Algorithm Implementation and Development

To develop and run these functions and algorithms, we used Python 3.12.8 in Visual Studios Code. For general numerical methods, we used NumPy [1]. For additional numerical methods, we used SciPy [4]. For PCA, Ridge Regression, KNN, and SVM, we used scikit-learn [3]. For plotting, we used Matplotlib [2].

First, we fit and transformed our training data into $k - PCA$ space, reconstructing our data for an increasing value of k until we could preserve 85% of it. We examined both specifically $k = 16$ and the k that preserved 85%. We visualized this data to make sure it was still reasonable.

Then, we developed an algorithm to isolate pairs of digits. We applied the Ridge Regression classifier on each pair to see how it performed given differing levels of similarity between digits.

The scikit-learn package contained a function `RidgeClassifierCV` with which we conducted cross-validation. This process involves both choosing different values of λ and subsetting the data, then running a Ridge Regression and evaluating how well it worked. This is based on minimizing MSE:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N} |f(\hat{X})_i - Y_i|^2$$

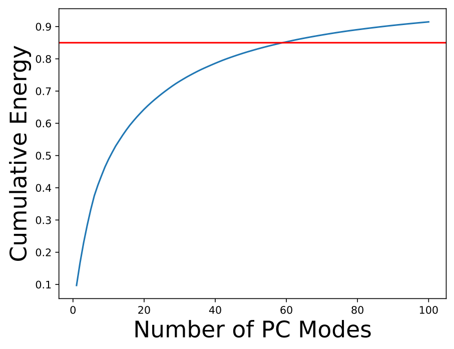
where $f(\hat{X})_i$ is the predicted value from our current model $\hat{f}$ and Y_i is the actual value, and N is the number of samples we have. By minimizing MSE, we optimize our λ from a predetermined options list. The cross-validation also yields a set of accuracies, with which we can calculate a mean and standard deviation for a more nuanced description of how the classifier is doing.

Once we were satisfied with how well this algorithm worked, we used the entirety of our training data to perform multi-class classification with Ridge, KNN, and SVM classifiers. We evaluated and compared each for accuracy.

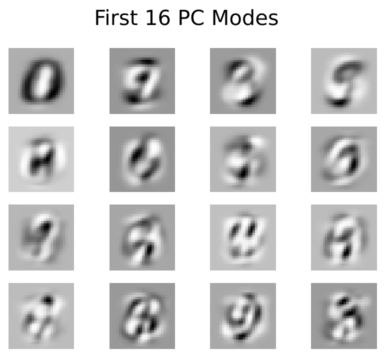
4. Computational Results

Note: We did not scale this particular data set, as it caused a much larger k value to preserve 85% of our original data set.

Our original data set was comprised of a variety of images of handwritten digits, the first 16 of which are displayed for reference in Figure 2a. Even from the first 16, we can see a significant variation in the way people draw their numbers. We reshaped these images into vectors to better perform PCA, displaying these results in Figure 1b. These modes show us the most prominent features from the data: the very first mode is strongly reminiscent of 0, which is a very consistent number despite an individual's particular handwriting. As we get further along in the PC modes to the less prominent features, it begins to look more and more chaotic.



(A) Cumulative energy

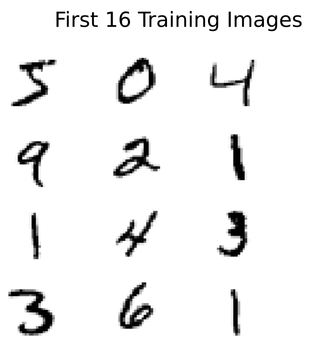


(B) $1-16-$PC

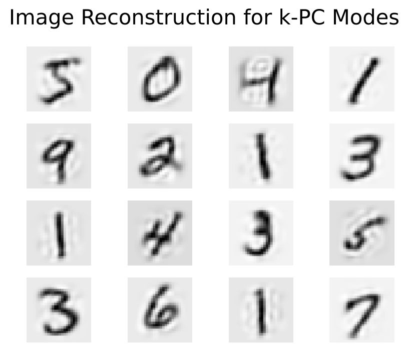
Figure 1. Different representations of various PC modes

These PC modes are quite helpful, though, in figuring out how many dimensions we really need to capture 85% of the energy from our original data set. As we increased the number of PC modes, we calculated the cumulative energy (3) for each. Then we inspected the increase, as shown in Figure 1a, until we hit the 85% threshold. This occurred at **k=59** PC modes.

To convince ourselves that this is an appropriate number of PC modes, we reconstructed the original images from $59-$PC modes and displayed them in Figure 2b for comparison against the original images in Figure 2a. Despite these reconstructed images being a bit more grainy and on a darker background, they accurately capture the most import aspects of the digit and can still be classified by eye. This will allow our future calculations to run much faster, as we are only considering 59 components instead of the original 786. Note that when this analysis was attempted with scaled data, we ended up with 116 modes needed to preserve 85% of the data, which is significantly larger and takes longer to run.



(A) Original images



(B) $1-16-$PC

FIGURE 2. First 16 handwritten digits in different contexts

We then found subsets of our handwritten data containing just two digits to compare against them directly against each other. We used just the Ridge classifier to compare $\{1, 8\}$, $\{3, 8\}$, and $\{2, 7\}$, and cross-validated our work so to ensure trust in our accuracy scores. These results are displayed in Table 1, where the maximum training accuracy would be 1.

Classifier	Digits	Training Accuracy	Training SD	Testing Accuracy
Ridge	1,8	0.9643	0.0027	0.9801
Ridge	3,8	0.9603	0.0081	0.9588
Ridge	2,7	0.9797	0.0014	0.9748
Ridge	All	0.8441	0.0098	0.8561
KNN	All	0.9738	0.0021	0.9893
SVM	All	0.9813	0.0016	0.9919

TABLE 1. Classifier accuracy scores in different contexts

Based on these results, we see that the testing accuracy is the highest for the pairs $\{1, 8\}$. This makes sense because these two digits are very different in their construction, especially in comparison to $\{3, 8\}$ and $\{2, 7\}$. The training accuracies, however, a lot more similar: the training accuracy for $\{2, 7\}$ ended up being the highest of the three. For this particular data set, this could make sense in that many people chose to add a loop to their 2, which helps distinguish it in shape from 7, despite the two digits looking very similar in text. $\{3, 8\}$ consistently had the lowest accuracy, which makes sense as the two digits are very similar no matter whether text or handwritten.

Despite these small differences, the accuracies for all pairs of digits are very similar. This could be because the training parameter λ was not as optimal as it could be. `RidgeClassifierCV` automatically evaluates an optimal λ, but it only selects from a set of options capped at $\lambda = 10$. Thus if there exists an optimal λ^* such that $\lambda^* > 10$, we will not ever find it.

Next, we attempted Ridge Regression on all of our data, as well as KNN and SVM. Similarly, Ridge Regression automatically optimized its λ. Our accuracy score was lower for all the digits together than the pairs we tested, but this makes sense as 10 digits are much harder to sort than just 2. Additionally, the classifier has to deal with other pairs of numbers like $\{3, 8\}$ that look similar, such as $\{5, 6\}$, with the added difficultly of these being mixed into the entire data set. This classifier ended up with lowest accuracy of the three.

Next we utilized a KNN regression, which yielded much better results. KNN is parameterized by k, a manual input of how many neighbors we want to look at to make our decision. We tested $k \in [1, 30]$ to ensure a wide variety, and saw what number of neighbors, k, would yield the best accuracy. These results are displayed in Figure 3.

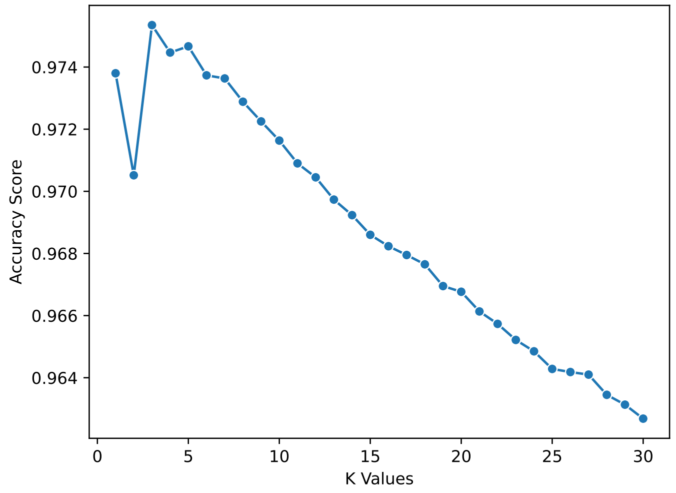


Figure 3. Accuracy as a function of k nearest neighbors

According to this analysis, $k = 3$ is the optimal value. Using this k, we found a much higher training and testing score than the Ridge Regression for all digits combined. But no matter the value of k, KNN yielded a better accuracy, which implies that our data is not well linearly separable. Thus a nonlinear classification method might work better.

So, for our final method, we used SVM with an RBF (Gaussian) kernel. This yielded the highest accuracy across all our methods for both training and testing data. We did not conduct a prior analysis to determine the best hyperparameters, in this case γ and C, and if we had done so would could have increased this value even further. The RBF kernel measures similarly between data points in higher dimensions, which makes it incredibly useful in our context even with lowered dimension data. It is also useful to model data that is nonlinear, which could be the case for the MNIST dataset and the Ridge Regression classifier did not perform as well.

While this method provided the highest accuracy, it also took the longest to run. Thus we must consider the trade-off between a fast runtime and slightly more accurate classification, especially between KNN and SVM.

Thus overall, we see that SVM performed the best, with KNN at a close second. Ridge Regression had both the worst accuracy and the largest standard deviation, meaning it performed differently across our different data sets, and might not be the best method to understand the interpret this particular data set. SVM is by far the most robust method with which to handle data that isn't easily linearly separable.

5. Summary and Conclusions

Over the course of this project, we evaluated three different classification algorithms for accuracy on a reduced-dimension data set. We now have several different classifiers at our disposal that can recognize handwritten digits from much fewer dimensions than we were originally given. Since we have cross-validated and evaluated these classifiers against each other, we have a high level of trust and confidence in our classifications. In the future, we may try to conduct a different type of PCA analysis, perhaps based on POD, to reduce the dimensions even further without sacrificing information. We could also extend it to further classification techniques to compare both speed and accuracy, such as Stochastic Gradient Descent.

Acknowledgements

The author is thankful to our TA Rohin Gilman for explanations of the Ridge and KNN classifiers. We also thank our peers Kaillah Selvaretnam, Heidi Neuman, Kate Everling, Quinn Kelly and Laura Pong for conceptual help surrounding the problem and cross-referencing algorithms and results.

References

[1] C. R. Harris, K. J. Millman, S. J. der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, SebastianBerg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020.

[2] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[4] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.